模型评价与模型改善

CS245　数据科学基础　陆朝俊
叶泽林 515030910468

1　问题描述

给定一个机器学习问题（如分类）以及相应的数据集，可以训练出一个初步的模型，但在投入实际应用前，需要对该模型的性能进行评价甚至改善。一般地，通过组合模型的方式，可以提高模型的效果。这次作业中，我将在自己建立的分类数据集上探索决策树模型单独工作以及组合工作时的性能。

2　解决方案[1]

2.1　数据集的建立及模型的选择

由于现有的中小型数据集存在一些难以满足实验要求的属性，我选择自己建立一个分类数据集 D（后文均以 D 表示此数据集），之后的所有实验均基于此数据集上进行。数据集 D 包含 500 个样本，30 个特征（存在冗余和重复的特征），为更好地看出模型性能的区别，我在 D 中添加了部分噪声。关于数据集 D 详细的设定参数可参考附录 A.1。

解决分类问题的机器学习模型主要有逻辑回归、决策树以及 KNN，在这次作业中，我选择决策树作为探索的模型。

2.2　探索模型单独工作的性能

选定数据集和模型后，我首先探索单独的决策树模型在数据集 D 上的性能，探索方法如下：

1. **直接划分训练集和测试集**

   在一般应用中，对模型性能最朴素的评价方法即为：将数据集划分为训练集和测试集，在训练集上训练模型后，根据模型在测试集上的表现评价模型性能。因此，我首先采用这个方法对模型的性能进行一个粗略的估计。

2. **分层交叉验证**

   从理论上说，仅凭在一个训练集 D_1 上训练出的模型 M_1 无法代表模型 M（在我的实验中为决策树模型）解决这个问题的效果。分层交叉验证正好解决了这一问题，其步骤大致如下：

   (a) 将数据随机等分为 k 个不相交子集 $D_1, D_2, ..., D_k$（每个子集的类分布与初始数据近似相同）；

   (b) 总共执行 k 次训练与测试，在第 i 折时，使用 D_i 作为测试集, 其他子集作为训练集；

   (c) 模型的性能由 k 次迭代的平均效果决定。

   分层交叉验证相当于在同一个数据集上进行了多次训练集-测试集划分。和简单的训练集-测试集划分相比，能更准确地评估模型来自样本的误差。

[1]本次作业的所有代码实现可参见附录 A.3

3. **网格搜索**

网格搜索是通过蛮力试验不同参数组合，从而选择最佳模型的方法。理论上来说，网格搜索与分层交叉验证相结合，可以找出模型解决给定问题最合适的参数，但这一过程需要花费大量的时间和计算资源，因此这次作业中我仅实现了一小部分参数空间上的网格搜索。

2.3 探索模型组合工作的性能

从一系列模型 $M_1, M_2, ..., M_k$ 创建组合模型 M^*，可以有效提高原模型的效果。组合方法主要分为 bagging 和 boosting 两种。Bagging 方法使用并行思想，M^* 通过整合各原模型的结果得到输出；Boosting 方法则采用串行思想，利用模型 M_{i-1} 的训练结果训练模型 M_i，直到 M_k。

3 结果展示

3.1 模型单独工作的性能

3.1.1 直接划分训练集和测试集

首先，我将原数据按照一定比例划分为训练集和测试集，划分时保证每一类型的数据在训练集和测试集的占比大致服从原分布。测试结果如表 1 所示。

表 1: 训练集-测试集划分探索模型在 D 上的性能

测试集占比	测试集上平均精度	测试集上平均召回率	测试集上平均 f1-score	训练集上平均 f1-score
0.1	0.59	0.56	0.56	1.00
0.2	0.57	0.56	0.56	1.00
0.3	0.57	0.56	0.56	1.00
0.4	0.56	0.56	0.55	1.00
0.5	0.50	0.48	0.48	1.00

测试结果表明：不论训练集-测试集的划分比例如何，该模型总是趋于过拟合，且在新数据上的效果较差。

3.1.2 分层交叉验证

上小节仅对决策树模型在 D 上单独工作的性能进行了粗略的估计，下面使用分层交叉验证对其性能做更完整的探索，结果参见图 1。

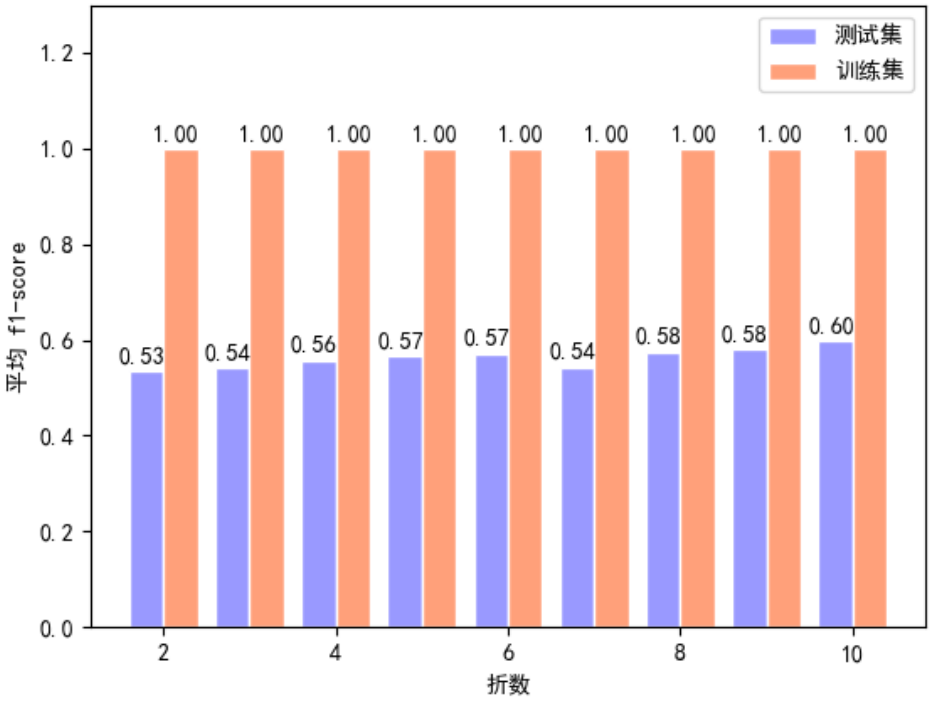

图 1: 分层交叉验证探索模型在 D 上的性能

3.1.3 网格搜索

这部分实验中，我结合网格搜索和分层交叉验证（10 折），得到决策树模型在 D 上的最佳效果。由于遍历所有参数需要消耗大量的时间和计算资源，我在实验中仅选择了模型的部分参数进行网格搜索。具体结果可参见表 2。

表 2: 网格搜索结果

参数名	参数含义	参数范围	模型效果最佳时的值
criterion			gini
max_depth	0.57		9
max_features	0.57		None
presort	0.56		True
splitter	0.50		best

经过对上表参数值的网格搜索，得到的模型在分层交叉验证中各测试集的平均 f1-score 为 0.61，这个结果在一定程度上代表了决策树模型在 D 上分类的最佳性能。

3.1.4 结果分析

通过上面三个小节的探索，可以得出结论：决策树模型单独工作时，在数据集 D 上分类的 f1-score 基本在 0.5-0.6 之间，最高仅达 0.61，且常趋于过拟合，可见该模型单独工作时效果较差。

A 附录

A.1 分类数据集 D 的详细信息

表 3: 分类数据集 D 的详细信息

属性	值
样本个数	50
特征个数	30
类个数	3
类分布	均匀分布
包含信息的特征占比	0.6
冗余特征的占比（含信息特征的线性组合）	0.1
重复特征的占比（随机取自含信息特征和冗余特征）	0.1
噪声占比	0.03

A.2 PCA 算法在主成分个数为 3 时的进一步可视化

A.3 主要代码

```
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']     # display Chinese
plt.rcParams['axes.unicode_minus'] = False       # display minus sign
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA

import os
import time

class bostonAnalyzer(object):
    def __init__(self):
        # load dataset
        self.dataset = datasets.load_boston()
        self.data = self.dataset.data
        self.target = self.dataset.target

        # for plot
        self.var = []
        self.t = []

        if not os.path.exists('report/img'):
            os.makedirs('report/img')

    def run(self, vis=False):
```

```
        '''
        Run PCA for 1-13 dimensions
        :param vis: True: plot dim 1-3; False: not plot
        '''

        self.var.clear()
        self.t.clear()

        with open('report/result.txt', 'w') as f:
            for i in range(13):
                t = time.time()
                pca_op = PCA(n_components=i)
                pca_res = pca_op.fit_transform(self.data)
                t = time.time() - t

                self.var.append(pca_op.explained_variance_ratio_.sum())
                self.t.append(t)

                # write log
                f.write('###### Dimension %d ######\n' % i)
                f.write(str(pca_res.shape) + '\n')
                f.write(str(pca_op.explained_variance_ratio_) + '\n')
                f.write(str(pca_op.explained_variance_ratio_.sum()) + '\n')
                f.write(str(t) + '\n\n')

                # visualize for debug & plot
                if vis:
                    if i == 1:
                        # plot 1 dimension
                        plt.scatter(pca_res[:, 0], pca_res[:, 0], s=14, c=self.target)
                        plt.savefig('report/img/pca-%d' % i)
                        plt.show()
                    elif i == 2:
                        # plot 2 dimensions
                        plt.scatter(pca_res[:,0], pca_res[:,1], s=8, c=self.target)
                        plt.savefig('report/img/pca-%d' % i)
                        plt.show()
                    elif i == 3:
                        # plot 3 dimensions
                        ax = plt.subplot(projection='3d')
                        ax.scatter(pca_res[:, 0], pca_res[:, 1], pca_res[:, 2], s=8, c=self.target)
                        plt.savefig('report/img/pca-%d' % i)
                        plt.show()
```

```
69
70      def show(self):
71          '''
72          Show the basic info of Boston dataset & plot k-lines
73          '''
74          print(self.data.shape)
75          print(self.target.shape)
76          self._k_line_radio()
77          self._k_line_time()
78
79      def _k_line_radio(self):
80          '''
81          Plot k-line of variant radio
82          '''
83          x = list(range(len(self.var)))
84          plt.scatter(x, self.var, s=14, c='r')
85          plt.plot(x, self.var)
86          plt.xlabel('主成分个数')
87          plt.ylabel('降维后各特征方差比例之和')
88          plt.savefig('report/img/kline-radio')
89          plt.show()
90
91      def _k_line_time(self):
92          '''
93          Plot k-line of time
94          '''
95          x = list(range(len(self.t)))
96          plt.scatter(x, self.t, s=14, c='r')
97          plt.plot(x, self.t)
98          plt.xlabel('主成分个数')
99          plt.ylabel('PCA算法消耗时间(s)')
100         plt.savefig('report/img/kline-time')
101         plt.show()
102
103 if __name__ == '__main__':
104     bA = bostonAnalyzer()
105     bA.run(True)
106     bA.show()
```